ALA-019 PATENT

# TCP/IP OFFLOAD DEVICE WITH FAST-PATH TCP ACK GENERATING AND TRANSMITTING MECHANISM

Clive M. Philbrick
Laurence B. Boucher
Stephen E.J. Blightman
Peter K. Craft
David A. Higgen
Daryl D. Starr

## CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/464,283, filed December 15, 1999, which in turn claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/439,603, filed November 12, 1999, which in turn: 1) claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/067,544, filed April 27, 1998, and 2) claims the benefit under 35 U.S.C. § 119(e)(1) of Provisional Application Serial No. 60/061,809, filed October 14, 1997.

[0002] This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/384,792, filed August 27, 1999, which in turn claims the benefit under 35 U.S.C. § 119(e)(1) of Provisional Application Serial Number 60/098,296, filed August 27, 1998.

[0003] This application also claims the benefit under 35 U.S.C. §120 of U.S. Patent Application Serial No. 09/802,426, filed March 9, 2001. The subject matter of all of the above-identified patent applications (including the subject matter in the Microfiche Appendix of U.S. Application Serial No. 09/464,283), and of the two above-identified provisional applications, is incorporated by reference herein.

CROSS REFERENCE TO COMPACT DISC APPENDIX

[0004] The Compact Disc Appendix, which is a part of the present disclosure, includes a recordable Compact Disc (CD-R) containing information that is part of the disclosure of the present patent document. A portion of the disclosure of this patent document contains material that is subject to copyright protection. All the material on the Compact Disc is hereby expressly incorporated by reference into the present application. The copyright owner of that material has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which:

[0006] Figure 1 is a diagram of a network data communication system in accordance with one embodiment of the present invention.

[0007] Figure 2 is a simplified block diagram of the intelligent network interface device (INID) of Figure 1.

[0008] Figure 3 is a diagram of a template header in accordance with an embodiment of the present invention.

[0009] Figure 4 is a section of code that executes on processor 28 of Figure 2. The code is for generating ACKs in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

[0010] Figure 1 is a diagram of one particular network system 1 in which the present invention operates. The present invention may, however, operate in numerous other systems. The system 1 of Figure 1 is set forth as one example only.

**[0011]** System 1 includes a first device 2 (in this case, a server) that both receives data from and transmits data to a second device 3 (in this case, a storage array) via a high-speed bidirectional full-duplex network link 4. Bidirectional network link 4 employs one particular transport layer protocol and one particular network layer protocol. In the present example, the transport layer protocol is the TCP protocol, the network layer protocol is the IP protocol, and bidirectional network link 4 is a gigabit ethernet network connection. Storage array 3 in this example includes a RAID controller 5 and a plurality of disc devices 6-9. Disc devices 6-9 are coupled via Small Computer System Interface (SCSI) links or Fibre Channel (FC) links 11-14 to the RAID controller. Relatively large amounts of network information pass over gigabit ethernet link 4 in accordance with the IP Storage protocol (also called "ISCSI") both from storage array 3 and to server 2 as well as from server 2 and to storage array 3.

**[0012]** Server 2 is coupled to gigabit link 4 by an intelligent network interface device (INID) 10. For additional information on INID 10, see U.S. Patent Application Serial No. 09/464,283 (the subject matter of which is incorporated herein by reference). Server 2 and INID 10 together perform protocol processing such that ISCSI communications received from ethernet link 4 may be processed by an ethernet MAC protocol processing layer, by an IP protocol processing layer, by a TCP protocol processing layer, and by an ISCSI protocol processing layer. Similarly, ISCSI communications output from server 2 onto ethernet link 4 may be processed by the ISCI protocol processing layer, by the TCP protocol processing layer, by the IP protocol processing layer, by the ethernet MAC protocol processing layer and are then output onto the ethernet link. For network communications where a particular set of transport layer, network layer, and MAC layer protocols is used, INID 10 accelerates network communication by using dedicated accelerator hardware on INID 10 to speed protocol processing. INID 10 is therefore said to perform TCP/IP offload functions for server 2.

**[0013]** In the present example, the particular set of protocols is the TCP protocol, the IP protocol, and the ethernet protocol. It is to be understood, however, that the particular set of transport, network, and MAC layer protocols may be different in another example. A packet whose transport layer protocol is the TCP protocol and whose network layer protocol is the IP protocol is called a TCP/IP packet. Generally speaking, the term "transport layer" as it is used here identifies that layer of the protocol stack that a TCP layer would be classified into, the term "network layer" as it is used here identifies that layer of the protocol stack that an IP layer would be classified into, and the term "MAC layer" as it is used here identifies that layer of the protocol stack directly beneath the network layer.

**[0014]** Figure 2 is a simplified block diagram that illustrates TCP/IP offloading performed by INID 10. A CPU 20 executes protocol stack software stored in memory 21. The protocol stack is illustrated as stack 22 involving a MAC layer, an IP layer, a TCP layer, and an ISCSI layer. It is to be understood that CPU 20 and stack 22 are part of INID 10 where the INID 10 is coupled to another processor of server 2, or alternatively CPU 20 and stack 22 are part of the remainder of server 2 to which INID 10 is coupled. In either case, stack 22 performs a significant amount (in some cases, substantially all) of the TCP/IP protocol processing under certain infrequently occurring and/or complex situations (this relatively slow protocol processing path is called the "slow-path), whereas the dedicated accelerator portion of INID 10 performs substantially all TCP/IP protocol processing under the remaining situations (this relatively fast protocol processing path is called the "fast-path"). Infrequently occurring situations include, in one particular example, situations in which TCP packets are received out of sequence.

**[0015]** In the embodiment of Figure 2, the dedicated accelerator hardware of INID 10 includes a physical layer interface 23, a DRAM 24, and a specially-designed integrated circuit 25. Integrated circuit 25 includes a receive sequencer 26, a transmit sequencer 27, a processor 28, a queue manager 29, MAC interface circuitry 30, an SRAM controller 31, an SRAM 32, a DRAM controller 33, and a bus interface 34. Processor 28 executes code stored in a control store portion of SRAM 32. Transmit sequencer 27 executes instructions that are hardcoded into integrated circuit 25. Processor 28 may, in certain embodiments, actually include three processors: a transmit processor, a receive processor, and a utility processor. For additional information on integrated circuit 25 and its operation, see U.S. Patent Application Serial No. 09/464,283 (the subject matter of which is incorporated herein by reference).

**[0016]** In the example of Figure 1, a data transfer of an ISCSI read reply from read target disc 6 occurs at the same time that a data transfer of an ISCSI write to write target disc 7 occurs. The ISCSI read reply causes data on read target disc 6 to pass across SCSI/FC link 11 to RAID controller 5 and across gigabit ethernet link 4 and into INID 10. The data passes over a first TCP/IP connection (denoted in Figure 1 as connection "1") that is established for this purpose. A flow of TCP packets (sometimes called "frames") is received onto INID 10 via physical layer interface (PHY) 23, MAC interface 30, and receive sequencer 26.

**[0017]** Receive sequencer 26 performs initial processing on each TCP packet, validates packet headers, generates a lookup hash from the TCP and IP headers of the packet, and determines whether the packet meets certain "fast-path candidate" criteria. For example, only packets having the particular set of transport layer protocol and network layer protocol may be handled via the fast-path. If the receive sequencer 26 determines from the headers of the packet that the packet does not fit these fast-path candidate criteria, then receive sequencer 26 sets an attention bit to a one indicating that the packet is not a "fast-path candidate". If on the other hand the packet fits the criteria, then receive sequencer 26 sets the attention bit to a zero indicating that the packet is a "fast-path candidate".

**[0018]** Receive processor 26 is called a "fly-by sequencer" because it analyzes the incoming packet on-the-fly as the packet is being received rather than putting the packet into a separate memory and then later retrieving the packet from the separate memory and performing the analysis using a more general purpose processor. The incoming packet information is placed into a buffer in DRAM 24 and a "receive descriptor" is pushed onto a "receive descriptor queue" (not shown) maintained by queue manager 29. The "receive descriptor" includes both the attention bit and a pointer (or set of pointers) that points to the buffer in DRAM 24.

**[0019]** Processor 28 pops the receive descriptor queue, obtains the pointer (or pointers) and the attention bit, and retrieves the associated packet information from DRAM 24. If the attention bit is set to a zero indicating that the packet is a "fast-path candidate", then processor 28 performs additional checking to determine whether the packet is in fact a "fast-path packet" that will be handled using the fast-path. There are a plurality of

TCP/IP connections, communications for which are handled by INID 10 using the fast-path. Processor 28 uses the hash made by receive sequencer 26 to determine if the TCP/IP connection of the packet is one of the TCP/IP connections handled in fast-path (for example, by identifying the packet with a context or a communication control block (CCB)). If the connection of the packet is determined to be one of the fast-path connections, then processor 28 performs the additional step of checking for one of numerous exception/error conditions that are not to be handled via the fast-path. One example of an exception condition is the packet having a TCP sequence number that is out of order. If processor 28 finds no exception/error condition, then the packet is determined to be a "fast-path packet". For additional information on the how a packet is determined to be a "fast-path packet" in one embodiment, see: 1) U.S. Patent Application Serial No. 09/464,283, and 2) U.S. Patent Application Serial No. 09/384,792 (the subject matter of these two applications is incorporated herein by reference).

[0020] Once the received packet is determined to be a "fast-path packet", processor 28 causes the data portion of the packet to be written into a destination in memory 21. In this way, the data portions of each successive TCP/IP packet of the ISCSI read reply is written into the destination so that the destination contains a single block of data without any TCP or IP headers. The ISCSI layer of stack 22 can then access the block of ISCSI data in memory 21 without CPU 20 and stack 22 having to perform TCP or IP protocol processing on the incoming TCP/IP packets.

[0021] At the same time that this fast-path ISCSI read is occurring, the data transfer of the ISCSI write is occurring. Data for the ISCSI write also passes through INID 10 via the fast-path. The file system of CPU 20 places data into a particular location in memory 21. This data is then transferred to DRAM 24 by a DMA controller (not shown) of integrated circuit 25. Processor 28 sections this data up and causes it to be transmitted onto ethernet link 4 as a plurality of TCP/IP packets. Processor 28 forms the header of each TCP/IP packet and merges that header with its data so that a complete packet exists in a buffer DRAM 24.

[0022] Processor 28 then pushes a pointer to that buffer onto a transmit queue P0TX maintained by queue manager 29. In accordance with one embodiment, there is a queue control register and a data register associated with transmit sequencer 27. The low priority queue P0TX is identified by a first identifying five-bit queue ID value present in a first five-bit field XmtQId of the queue control register (XmtCfg). The higher priority queue P1TX is identified by a second identifying five-bit queue ID value present in a second five-bit field PriQId of the queue control register (XmtCfg). Use of the two priority transmit queues is enabled by writing a one into a one-bit field PriQEn of the queue control register (XmtCfg). To accomplish the push, processor 28 first writes an appropriate value to the queue control register (XmtCfg) and then writes the pointer to the queue data register. The second write to the queue data register actually causes the push.

[0023] Transmit sequencer 27 then pops the transmit queue P0TX, obtains the pointer, uses the pointer to read the packet information from DRAM 24, and then transmits the packet information from the INID as a TCP/IP packet via MAC 30 and PHY 23. The TCP/IP packets associated with the ISCSI write are communicated via a second TCP/IP connection that is denoted "0" in Figure 1.

[0024] INID 10 transmits TCP/IP packets onto gigabit ethernet link 4 at close to the theoretical unidirectional maximum throughput rate of gigabit ethernet link 4 (1000 megabits per second). In the present example, there are times that INID 10 is not the limiting factor in maintaining throughput rate, and TCP/IP packets for transmission begin to back up in INID 10. Pointers to various TCP/IP packets in DRAM 24 to be transmitted therefore begin to back up in transmit queue P0TX.

[0025] Not only does INID 10 output TCP/IP packets at close to the theoretical maximum unidirectional throughput rate of ethernet link 4, but INID 10 also receives TCP/IP packets from link 4 at close to the theoretical unidirectional throughput rate (1000 megabits per second). Under typical conditions, the TCP protocol requires that a receiving device return an acknowledge packet (ACK) back to the transmitting device for every other received TCP packet. The use of ACKs and their associated windows in the TCP protocol prevents the overloading of a receiving device by stopping the transmitting device until the receiving device has caught up.

[0026] In the present example where transmission across link 4 into INID 10 occurs at close to the theoretical 1000 megabits per second limit for link 4, transmission across link 4 into INID 10 should not be stopped due to INID 10 not sending back an ACK for a

TCP packet received. If ACKs for the incoming data stream of connection "1" were transmitted in normal course via transmit queue P0TX, then pointers to the ACKs for connection "1" would get pushed onto the P0TX transmit queue after pointers to the backed up outbound TCP packets for connection "0".

[0027] TCP ACKs associated with the ISCSI read are therefore generated and transmitted with priority using the fast-path. A fast-path ACK for the connection "1" ISCSI read is generated without having to pass the TCP ACK down to a discrete IP protocol layer, and then down to a discrete MAC protocol layer. Such sequential processing of the TCP ACK through a sequence of discrete transport and network protocol processing layers may be slowed because a lower level processing layer (for example, the IP layer) to which the ACK is passed may not be provisioned to give processing priority to the ACK. Accelerating processing through a single layer of protocol processing may therefore not result in the ACK being processed through all the layers with priority. The ACK may therefore not be output from INID 10 as rapidly as is desired.

[0028] To avoid such potential sequential protocol processing delay issues, INID 10 employs a template header 35 and a software finite state machine 36 executing on processor 28.

[0029] Figure 3 is a diagram of template header 35. Although considerable detail is set forth here with respect to template header 35, it is to be understood that Figure 3 sets forth but one example of a template header in accordance with an embodiment of the present invention. Other template header formats can be employed in accordance with the present invention.

[0030] Figure 4 is a listing of code for software finite state machine (FSM) 36. In the code, "Q_XMTPRI1C" designates a high-priority transmit queue used for ACKs (another lower-priority transmit queue is used to transmit ordinary data packets). The structure and operation of the FSM is evident from the code itself. The functions performed by the various instructions in the code are generally evident from the names of the instructions themselves. For additional information on the instructions, see the description of the instruction set found in Microfiche Appendix B of U.S. Patent Application Serial No. 09/464,283 (the subject matter of which is incorporated herein by reference).

[0031] Finite state machine 36 covers both TCP and IP protocol processing and thereby "flattens" protocol processing such that the TCP ACK is generated without having to wait for sequential protocol processing of various protocol processing layers. ACKs generated are handed in a fairly direct manner to the hardware interfacing to the physical ethernet link without having to go through separate lower level IP and MAC processing layers.

[0032] Finite state machine 36 reads template header 35 from SRAM 32. Template header 35 includes various TCP and IP fields that processor 28 and finite state machine 36 fill in. In one embodiment (see Figure 3), these TCP and IP fields include a sixteen-bit IP Identification field, a sixteen-bit IP header checksum field (where the IP header includes the IP ID), a sixteen-bit IP total length field that indicates the total length of the IP datagram, a thirty-two bit TCP sequence number field, a thirty-two bit ACK number field, a sixteen-bit TCP window size field, a twelve-bit TCP flag field, and a sixteen-bit TCP checksum field. Finite state machine 36 uses information about the connection and the packet received (such as, for example, the TCP source and destination ports and the IP source and destination addresses) to fill in the TCP and IP fields in template header 35. Template header 35 as stored in SRAM 32 may contain partial checksums and/or partial length values that processor 28 uses to determine final checksum values and/or final length values that are filled in. In some embodiments, template header 35 as stored in SRAM 32 already includes the correct TCP source and destination ports and IP source and destination addresses.

[0033] Once processor 28 and finite state machine 36 have filled in the TCP and IP fields in the template header, the filled-in template header is transferred to a buffer in DRAM 24. If the ACK is to include a data payload, then the data payload is preloaded into the buffer such that the writing of the template header into the buffer results in the template header being prepended to the front of the data payload. An ACK can, for example, be incorporated into a data packet provided that the data packet is to be transmitted to the appropriate port and address (this is called piggy-backing).

[0034] In the case of the fast-path TCP ACK generated in the present example, however, there is no data payload. Once the TCP ACK is in the buffer, processor 28 causes a pointer (or a set of pointers) to the buffer to be pushed onto a high priority transmit queue P1TX. Transmit sequencer 27 is designed to pop a pointer off higher priority transmit queue P1TX in preference to popping a pointer off lower priority transmit queue P0TX. Accordingly, transmit sequencer 27 pops the ACK pointer (or the set of pointers) for connection "1" off transmit queue P1TX before it pops pointers off transmit queue P0TX for outbound connection "0" data. The fast-path TCP ACK for connection "1" is transmitted from INID 10 before any of the backed-up TCP data packets for connection "0".

[0035] In this example, the TCP ACK is output from INID 10 before any of the associated data from read target disc 6 is transferred from INID 10 and into the destination in memory 21. (This need not be the case, however. Associated data or some portion of the associated data may be transferred from INID 10 and into the destination in memory 21 before the TCP ACK is output from INID 10.) In the particular example described above, the rate of information transmitted onto ethernet link 4 from INID 10 remains substantially the same but the TCP ACK is transmitted back to the transmitting RAID device earlier, thereby decreasing the amount of time that RAID controller 5 is stopped from transmitting due to the RAID controller 5 not having received connection "1" ACKs. In this example, an ACK is approximately 64 bytes long, whereas a typical data packet is approximately 1500 bytes long. An INID in accordance with an embodiment of the present invention was tested in a Tolly Chariot bidirectional performance test. The INID achieved a 90 megabits per second throughput in both directions on a full-duplex 100 megabits/second link (in a first direction into the INID from the network as well as in a second direction out of the INID and to the network). Bandwidth is maximized and latency is minimized.

[0036] In some embodiments, the output of TCP/IP packets is further accelerated by not filling in a checksum field before the TCP/IP packet is loaded into a DRAM buffer, but rather the final checksum is merged into the TCP/IP packet as the packet is being output from INID 10 in accordance with methods set forth in U.S. Patent Application Serial No. 09/802,426 (the subject matter of which is incorporated herein by reference). In some embodiments, a destination in memory 21 where the data for the ISCSI read is placed is determined in accordance with methods set forth in U.S. Patent Application Serial No. 09/789,366 (the subject matter of which is incorporated herein by reference). Control of a TCP/IP connection can be passed by numerous techniques including, for example, moving a communication control block (CCB) for the connection to the device that is to assume control of the connection, or by maintaining a CCB for the connection in one location and merely transferring ownership of the connection by setting an ownership bit that identifies which device is in control of the CCB and the TCP/IP connection. Techniques set forth in U.S. Patent Application Serial No. 09/855,979 (the subject matter of which is incorporated herein by reference) can be practiced in combination with the TCP ACK generating and transmitting mechanisms set forth above. Techniques set forth in U.S. Patent Application Serial No. 09/970,124 (the subject matter of which is incorporated herein by reference) can be used to facilitate the executing of solicited session layer read commands (for example, ISCSI read commands) in combination with the TCP ACK generating and transmitting mechanisms set forth above.

[0037] The CD Appendix includes the following: 1) folder CPU, 2) folder XCV, 3) folder INCLUDE, 4) file fsma.txt, and 5) file fsms.txt. Folder CPU contains a hardware description of processor 28 in verilog. Folder XCV contains files that start with "xmt" and files that start with "rcv". The "xmt" files are a hardware description of transmit sequencer 27. The "rcv" files are a hardware description of a receive processor that in some embodiments is used in place of and performs the functions of receive sequencer 26. The file that ends in ".mal" is a program of instructions executed by the receive processor described by the verilog code. Folder INCLUDE is a definition file for the verilog code in folders CPU and XCV. The files fsma.txt and fsms.txt together are code for the finite state machine that executes on processor 28. A hardware description in verilog of queue manager 29 is found in U.S. Patent Application Serial No. 09/416,925 (the subject matter of which is incorporated herein by reference).

[0038] INID 10 may include an input priority mechanism whereby incoming ACKs are receive processed on a higher priority basis than are ordinary incoming data packets. In one embodiment, incoming ACKs are marked for high priority processing by setting three priority 802.1P bits in the VLAN tag header extension in the MAC header of the TCP ACK. The receive sequencer uses these bits to identify the high priority ACK such that a buffer descriptor to the high priority ACK is put onto a high priority receive queue, whereas buffer descriptors for other ordinary incoming packets are put onto a lower priority receive queue.

[0039] Although certain specific exemplary embodiments are described above in order to illustrate the invention, the invention is not limited to the specific embodiments. The above-described methods of generating fast-path TCP ACKs are more generally applicable to generating fast-path responses back to transmitting devices. INID 10 can be part of a memory controller integrated circuit or an input/output (I/O) integrated circuit or a bridge integrated circuit of a microprocessor chip-set. In some embodiments, the network interface device is part of an I/O integrated circuit chip such as, for example, the Intel 82801 integrated circuit of the Intel 820 chip set. INID 10 may be integrated into the Intel 82815 Graphics and Memory Controller Hub, the Intel 440BX chipset, or the Apollo VT8501 MVP4 North Bridge chip. Multiple template headers can be used, one for each particular set of transport, network, and MAC protocols that is fast-path processed. Hardcoded receive sequencer 26 is, in some embodiments, replaced with a receive processor that executes instructions stored in static RAM or other suitable memory. The instructions executed by the receive processor are, in some embodiments, downloaded upon power-up of INID 10 into the memory on INID 10, thereby facilitating the periodic updating of receive processor functionality. The high and low priority transmit queues set forth above may be implemented in software or may be implemented in more specialized queue hardware. A hardware queue manager may or may not be used. Similarly, hardcoded transmit sequencer 27 is, in some embodiments, replaced with a transmit processor that executes instructions stored in static RAM or other suitable memory. Accordingly, various modifications, adaptations, and combinations of various features of the described embodiments can be practiced without departing from the scope of the invention as set forth in the following claims.